PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 12/02 — A2

(11) International Publication Number: **WO 00/41079**

(43) International Publication Date: 13 July 2000 (13.07.00)

(21) International Application Number: PCT/EP99/10301

(22) International Filing Date: 20 December 1999 (20.12.99)

(30) Priority Data:
99200015.8 6 January 1999 (06.01.99) EP

(71) Applicant: KONINKLIJKE PHILIPS ELECTRONICS N.V. [NL/NL]; Groenewoudseweg 1, NL–5621 BA Eindhoven (NL).

(72) Inventors: SOEPENBERG, Gerrit, H.; Prof. Holstlaan 6, NL–5656 AA Eindhoven (NL). VUGTS, Johannes, A., G.; Prof. Holstlaan 6, NL–5656 AA Eindhoven (NL).

(74) Agent: DEGUELLE, Wilhelmus, H., G.; Internationaal Octrooibureau B.V., Prof. Holstlaan 6, NL–5656 AA Eindhoven (NL).

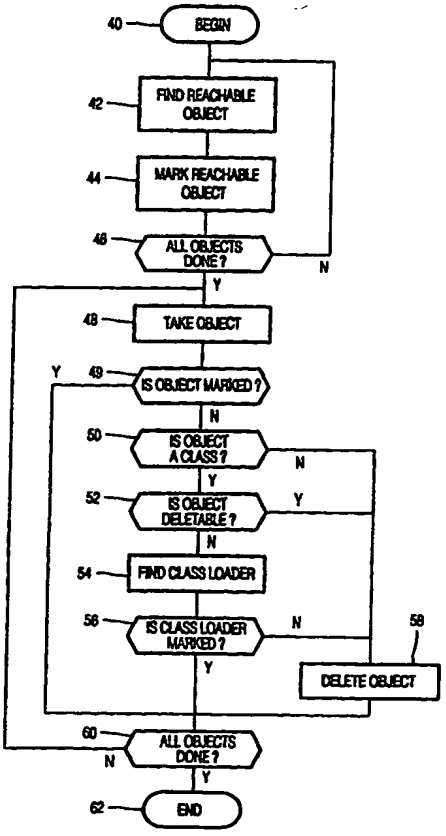
(81) Designated States: BR, CA, CN, JP, KR, MX, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: ARRANGEMENT FOR EXECUTING PROGRAM CODE WITH REDUCED MEMORY REQUIREMENTS

(57) Abstract

A Java virtual machine (2), comprises an execution engine for executing Java byte code and a memory management system (14). The memory management system (14) performs garbage collection to remove objects that are not reachable from the current state of the Java program. In the prior art memory management system objects of the representing a class definition are only removed from memory if the classloader object which has loaded the class object is not reachable. According to the present invention the memory management system (14) is arranged for removing classes which do not have any instantiated objects which can be reached in the present state of the program even if the classloader is still reachable. This results in a substantial saving of memory. Preferably the objects representing a class definition have a method which can signal to it that the definition can be deleted from memory if it has no instantiated objects anymore.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

Arrangement for executing program code with reduced memory requirements.

The present invention relates to an arrangement for executing program code, said arrangement comprises definition loading means for loading at least two definition entities into memory, said definition entities being associated with said definition loading means, the arrangement further comprises object creating means for creating objects according to said definition entity.

The present invention is also related to a method for executing program code and a tangible medium carrying a computer program for performing a method for executing program code.

An arrangement according to the preamble is known from the publication "The Java™ Virtual Machine Specification" by Tim Lindholm and Frank Yellin, which is available from the Sun web site and is converted from the printed book, September 1996, first printing.

In recent years the popularity of the Java platform has increased substantially. Java is a platform which comprises the Java language which is compiled to a byte code which runs on a Java Virtual Machine.

The use of a byte code which runs on a virtual machine has the advantage that the compiled byte code can be executed on every combination of hardware and operating system if a virtual machine exist for such combination.

The Java language is object oriented, which means that during the execution of the programs objects are created according to a definition entity, which can be a class or an interface in the Java language. Before objects can be created, the needed definition entities (classes or interfaces) have to be loaded first in memory. This loading is done by the definition loading means, which are called class loaders in the Java language. There can exist different definition loading means for different ways of loading definition entities. In Java there can e.g. be different class loaders for loading classes from disk, from a TCP/IP based network or loading them from an MPEG-2 Transport stream.

The Java virtual machine has a memory management system which performs garbage collection to delete automatically objects and classes that are not needed anymore.

When Java is to be used in consumer appliances such as TV's and set-top boxes, only limited amounts of memory resources are available. Currently the use of Java for consumer applications is quite difficult due to that limited amount of memory available.

The object of the present invention is to provide an arrangement according to the preamble in which the required amount of memory is decreased.

To achieve said objective, the arrangement according to the invention is characterized in that the arrangement comprises memory management means for removing from memory definition entities having no related objects, said definition loading means being associated with definition entities still having related objects.

By removing definition entities having no related objects anymore from memory, substantial amounts of memory can be freed, resulting in that the arrangement requires less memory than the arrangement according to the prior art.

The invention is based on the recognition that in the present specification of the Java virtual machine it is prescribed that a definition entity may only be deleted if its definition loading means is not reachable anymore. This means that no objects exist anymore of which the definition entity has been loaded by said definition loading means.

This will be made clearer in the following example. If a class loader (definition loading means) has loaded two classes (definition entities), a first loaded class having no related object anymore may not be removed from memory as long the other class still has related objects. The result of this is that classes remain unnecessary long in memory.

By using the inventive idea according to the present invention, classes may be unloaded as soon they are not needed anymore.

An embodiment of the present invention is characterized in that the arrangement comprises designation means for designating definition entities as removable, in that the memory management means are arranged for removing from memory definition entities having no related objects when the definition entity is designated as removable, and in that the memory management means are arranged for removing from memory definition entities corresponding to definition loading means being associated with only definition entities having no related objects.

There may be circumstances that it is undesirable that definition entities having no related objects are always removed from memory. This can be the case when class variables

are modified by the execution of class methods. If such a class is removed from memory and is reloaded when it is needed again, the value of the class variable may be changed.

By only removing definition entities without related objects when they are designated as removable, the application program can control this removal by designating a definition entity as removable or not removable.

A further embodiment of the invention is characterized in that said arrangement comprises storing means for storing an identification of definition entities that are designated as removable.

An easy way of communicating to the memory management means whether a class can be unloaded if it has no related objects is to store a table in which all loaded classed are entered together with an indication whether this class is removable or not. Alternatively, it is possible to store the identification in an object header associated with each class object.

The present invention will now be explained with reference to the drawings.

Fig. 1 shows the architecture of a Java virtual machine in which the present invention can be used.

Fig. 2 shows a class loader with related classes and instantiated objects.

Fig. 3 shows two class loaders, one having a class with an instantiated object.

Fig. 4 shows a flow diagram of a program according to the invention for use in the memory management system 14 according to Fig. 1.

In the Java virtual machine according to Fig. 1, a class management subsystem 6 is arranged for loading classes not present in memory from a local hard drive or from a network.

The classes to be loaded by the class management subsystem 6 are first checked by a code verifier 4. Normally only the classes loaded from a network are verified because they can be from an unknown, less reliable source. At loading the classes a syntax check is performed. Also the "semantic" consistency of the loaded class is checked. This includes e.g. the checking whether the constants defined in a class are consistent with their type. It is e.g. checked whether constants defined as strings are indeed strings.

During runtime a byte code verification is performed. This includes checking of consistency of the stack, i.e. verifying that no stack overflow and underflow occurs. It also

includes checking whether the type of data put into variables corresponds to the declaration of said variables.

An execution engine 8 is arranged for executing the verified byte code. For this execution the execution engine has three subsystems available, i.e. a system call subsystem 8 for invoking the operating system to perform certain tasks, such as the loading of an actual file from disk or from a network, or displaying graphical elements on a display device. Furthermore the Virtual Machine comprises a thread system 12 used for creating separate threads. The thread sub system 12 is also arranged for synchronizing separate threads.

The Java VM also comprises a memory management subsystem 14 which includes a garbage collector for automatically removing obsolete items from memory. In the Sun implementation of the Java VM a so-called "mark and sweep" garbage collection algorithm is used. This type of garbage collection algorithm is well known to those skilled in the art. In this algorithm, each object present in the heap has a so-called mark bit. The process of garbage collection starts with resetting the mark bit for all objects present in the heap. Subsequently all fields and variables are examined for references to objects in the heap. For each object in the heap that is referenced by a field or variable, the mark bit is set. Finally the garbage collector sweeps the heap and reclaims the memory area used by objects of which the mark bit is not set.

Finally, the Java VM comprises a Native Method Interface subsystem 16, enabling the use of functions written in C, C++ or assembler. This may be convenient in order to reuse existing libraries or to be able to use optimized assembler routines to improve performance.

In the diagram according to Fig. 2, objects present in the heap of the Java VM according to Fig. 1 are presented. Fig. 2 shows a first class 22 and a second class 20, which are loaded by a class loader 18. From class 22 one object 24 is instantiated and class 20 has no instantiated objects.

When using the garbage collection algorithm according to the prior art, none of the objects 18, 20, 22 and 24 will be garbage collected, because the classloader 18 is still reachable (having an object 24 instantiated from a class loaded by said classloader 18). In the arrangement according to the invention, the class 20 can be removed from memory because it has no instantiated objects. Preferably, a method is added to the class Class to indicate whether a class can be unloaded when it has no instantiated objects anymore. The syntax of such a method can be "EnableClassUnloading (class)". This method can set a bit in the classheader of the relevant class indicating that this class is deletable. Alternatively it is possible that a table is constructed in the VM in which all deletable classes are registered.

In Fig. 3 another group of objects present in the heap of a Java VM according to Fig. 1 is represented. Fig. 3 shows a first classloader 36 that has loaded a first class 32 and a second class 34. None of the classes 32 and 34 is reachable. These classes 32 and 34 and their classloader 36 will be removed from memory irrespective whether the method EnableClassUnloading is called or not, because the class loader 36 is not reachable.

Fig. 3 shows also a second classloader 26 which has loaded a class 28 from which an object 30 is instantiated. The class 28 has one reachable object 30. Consequently the classloader 26, the class 28 and the object 30 will not be unloaded (removed from memory).

It is observed that in the Java 1.2 SDK the concept of reference objects was introduced, resulting in objects that can be strongly and weakly reachable. The prior art garbage collector removes all objects that are not strongly reachable. It is observed that, when reference objects are used, the concept reachable as used in the above explanations with reference to Figures 2 and 3 means strongly reachable.

In the flowgraph according to Fig. 4, the numbered instructions have the following meaning:

No.	Inscription	Meaning
40	BEGIN	The execution of the garbage collector is started.
42	FIND REACHABLE OBJECT	A reachable object is found.
44	MARK REACHABLE OBJECT	The object is marked as reachable.
46	ALL OBJECTS DONE ?	It is checked whether all objects have been examined.
48	TAKE OBJECT	A new object is taken for garbage collection.
49	IS OBJECT MARKED ?	It is checked whether the object was marked.
50	IS OBJECT A CLASS ?	It is checked whether the object is a class.
52	IS OBJECT DELETABLE ?	It is checked whether the object is deletable.
54	FIND CLASS LOADER	The class loader corresponding to the current object is located.
56	IS CLASS LOADER MARKED ?	It is checked whether the class loader was marked.
58	DELETE OBJECT	The object is deleted from memory.
60	ALL OBJECTS DONE ?	It is checked whether all objects are processed.
62	END	The execution of the garbage collector is terminated.

In instruction 40 of the program according to Fig. 4 the program is started and the necessary initializations take place. In instruction 42, starting from the fields and variables corresponding to the current state of the program, reachable objects are searched. If a reachable object is found, the object is marked in instruction 44. In instruction 46 it is checked whether all reachable objects are found. This is the case when all fields and variables in the current state of the VM have been traced for corresponding objects. If not all reachable objects are found yet the program jumps to instruction 42 for finding the next reachable object.

If all possible reachable objects are found, the "mark phase" of the "mark and sweep" garbage collection algorithm is completed, and the "sweep phase" is started.

In instruction 48 an object from the heap is selected for examination. In instruction 49 it is checked whether the object is marked. If the object is marked, it must not be deleted from memory, and the program continues at instruction 60. If the object is not marked, in instruction 50 it is checked whether the object is a class. This can e.g. be determined from a field in an object header associated with the object. If the object is not a class, the program continues with instruction 58 in which the object is deleted from memory. Subsequently the program is continued at instruction 60.

If the object is a class, in instruction 52 it is checked whether the class is designated as deletable. This can e.g. be done by checking whether a "deletable bit" in the object header has been set by invoking the EnableClassUnloading method. Alternatively, it is also possible that a table containing all deletable classes is consulted. If the object is deletable, in instruction 58 the class object is deleted in instruction 58.

If the class is not marked as deletable, in instruction 54 the class loader which loaded the class is searched. This can be done by reading a table in the VM which comprises the loaded classes and their associated class loader. After having found the class loader, it is checked in instruction 56 whether the class loader object is marked. If the class loader is not marked, in instruction 58 the class object is deleted. If the class loader is marked, the class is not deleted from memory and the program is continued at instruction 60.

In instruction 60 it is checked whether all objects in the heap have been processed. If this is the case, the program is terminated in instruction 62. Otherwise the program continues with instruction 48 to process the next object in the heap.

CLAIMS:

1. Arrangement for executing program code, said arrangement comprises definition loading means for loading at least two definition entities into memory, said definition entities being associated with said definition loading means, the arrangement further comprises object creating means for creating objects according to said definition entity, characterized in that the arrangement comprises memory management means for removing from memory definition entities having no related objects, said definition loading means being associated with definition entities still having related objects.

2. Arrangement according to claim 1, characterized in that the arrangement comprises designation means for designating definition entities as removable, in that the memory management means are arranged for removing from memory definition entities having no related objects when the definition entity is designated as removable, and in that the memory management means are arranged for removing from memory definition entities corresponding to definition loading means being associated with only definition entities having no related objects.

3. Arrangement according to claim 2, characterized in that said arrangement comprises storing means for storing an identification of definition entities that are designated as removable.

4. Method for executing program code, said method comprises loading at least two definition entities into memory by a definition loading entity, said definition entity being associated to the definition loading entity, the method further comprises creating objects according to said definition entity, characterized in that the method comprises removing from memory definition entities having no related objects, said definition loading entity being associated with definition entities still having related objects.

5. Method according to claim 4, characterized in that the method comprises designation of definition entities as removable, removing from memory definition entities having no

related objects when the definition entity is designated as removable, and in that the method comprises removing from memory definition entities associated with only definition entities having no related objects.

6. Method according to claim 5, characterized in that said method comprises storing an identification of definition entities that are designated as removable.

7. Tangible medium carrying a computer program for performing a method for executing program code, said method comprises loading at least two definition entities into memory by a definition loading entity, said definition entity being associated to the definition loading entity, the method further comprises creating objects according to said definition entity, characterized in that the method comprises removing from memory, definition entities having no related objects, said definition loading means being associated with definition entities still having related objects.

8. Tangible medium according to claim 7, characterized in that the method comprises designating definition entities as removable, removing from memory definition entities having no related objects when the definition entity is designated as removable, and in that the method comprises removing from memory definition entities associated with only definition entities having no related objects.

9. Tangible medium according to claim 8, characterized in that said method comprises storing an identification of definition entities that are designated as removable.

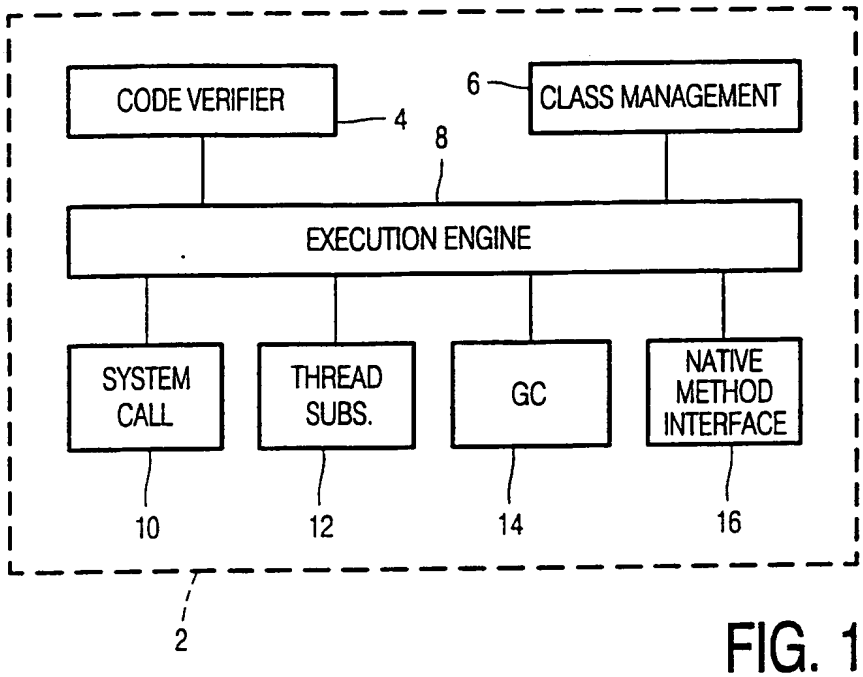


FIG. 1

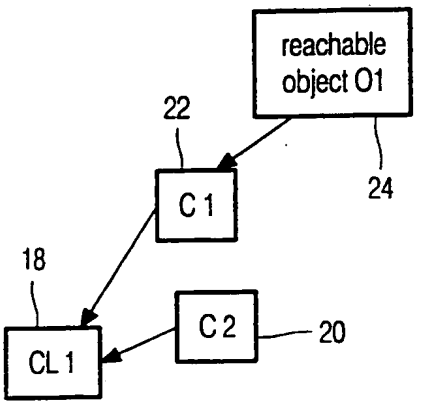


FIG. 2

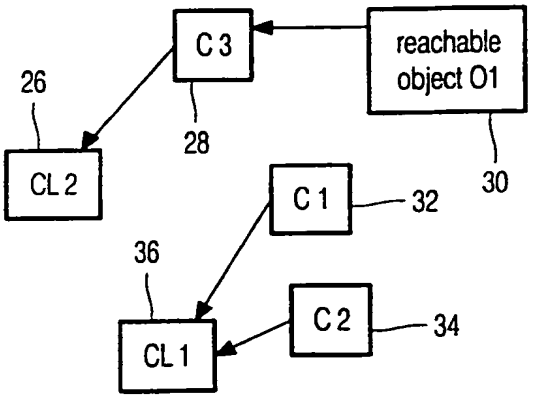


FIG. 3

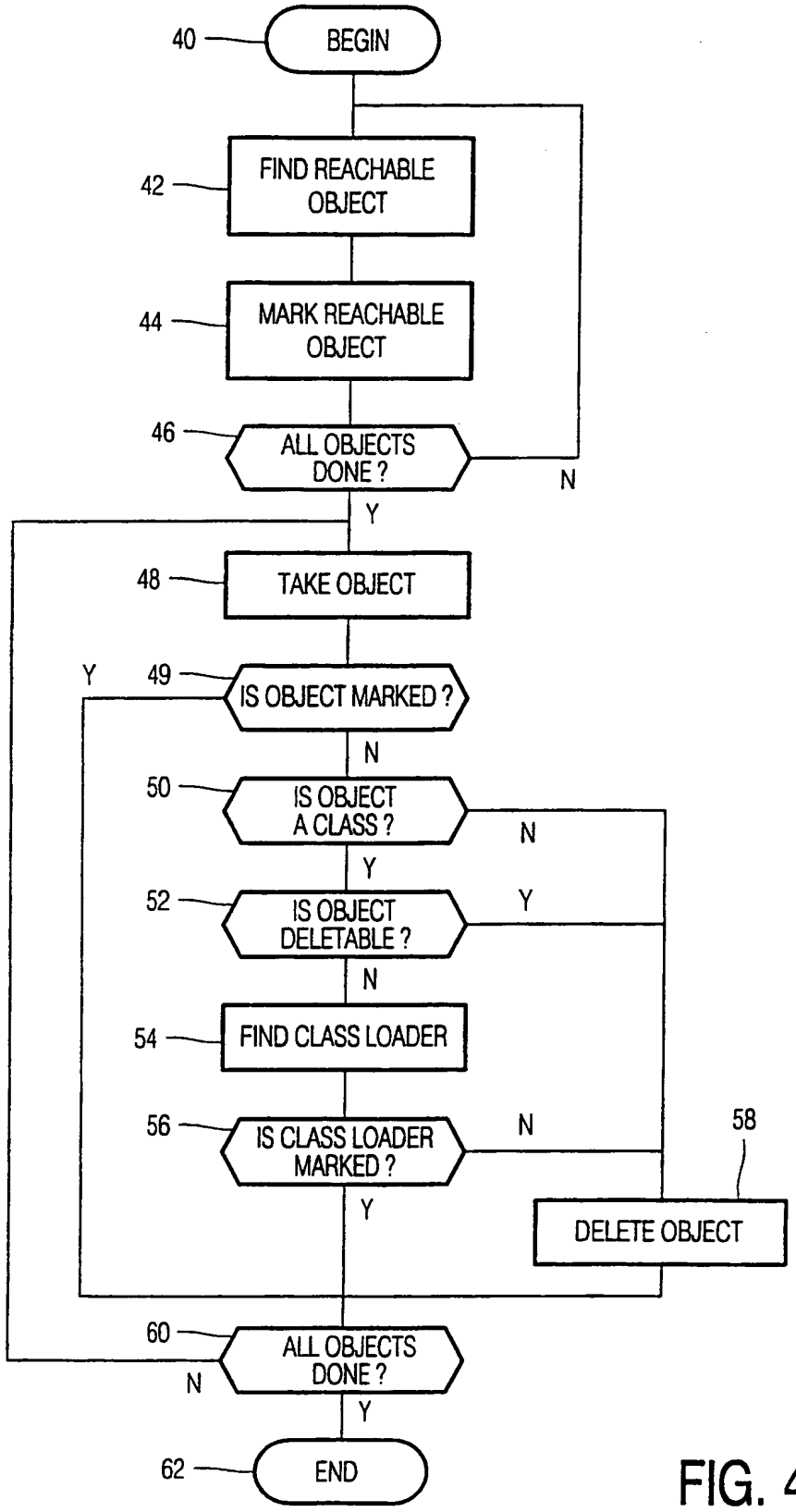

40
BEGIN
42
FIND REACHABLE OBJECT
44
MARK REACHABLE OBJECT
46
ALL OBJECTS DONE ?
N
Y
48
TAKE OBJECT
Y
49
IS OBJECT MARKED ?
N
50
IS OBJECT A CLASS ?
N
Y
52
IS OBJECT DELETABLE ?
Y
N
54
FIND CLASS LOADER
56
IS CLASS LOADER MARKED ?
N
58
Y
DELETE OBJECT
60
ALL OBJECTS DONE ?
N
Y
62
END

FIG. 4

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 12/02, 9/44, 9/445	A3	**(11) International Publication Number:**	**WO 00/41079**
		(43) International Publication Date:	13 July 2000 (13.07.00)

(21) International Application Number: PCT/EP99/10301

(22) International Filing Date: 20 December 1999 (20.12.99)

(30) Priority Data:
99200015.8 6 January 1999 (06.01.99) EP

(71) Applicant: KONINKLIJKE PHILIPS ELECTRONICS N.V. [NL/NL]; Groenewoudseweg 1, NL–5621 BA Eindhoven (NL).

(72) Inventors: SOEPENBERG, Gerrit, H.; Prof. Holstlaan 6, NL–5656 AA Eindhoven (NL). VUGTS, Johannes, A., G.; Prof. Holstlaan 6, NL–5656 AA Eindhoven (NL).

(74) Agent: DEGUELLE, Wilhelmus, H., G.; Internationaal Octrooibureau B.V., Prof. Holstlaan 6, NL–5656 AA Eindhoven (NL).

(81) Designated States: BR, CA, CN, JP, KR, MX, European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

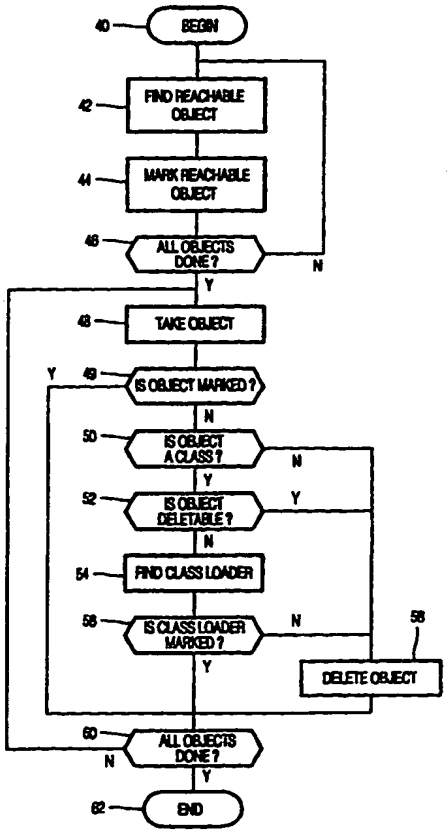
Published
With international search report.

(88) Date of publication of the international search report:
2 November 2000 (02.11.00)

(54) Title: ARRANGEMENT FOR EXECUTING PROGRAM CODE WITH REDUCED MEMORY REQUIREMENTS

(57) Abstract

A Java virtual machine (2), comprises an execution engine for executing Java byte code and a memory management system (14). The memory management system (14) performs garbage collection to remove objects that are not reachable from the current state of the Java program. In the prior art memory management system objects of the representing a class definition are only removed from memory if the classloader object which has loaded the class object is not reachable. According to the present invention the memory management system (14) is arranged for removing classes which do not have any instantiated objects which can be reached in the present state of the program even if the classloader is still reachable. This results in a substantial saving of memory. Preferably the objects representing a class definition have a method which can signal to it that the definition can be deleted from memory if it has no instantiated objects anymore.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

INTERNATIONAL SEARCH REPORT

Intern nal Application No
PCT/EP 99/10301

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F12/02 G06F9/44 G06F9/445

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	SUN MICROSYSTEMS, INC.: CLARIFICATIONS AND AMENDMENTS TO THE JAVA LANGUAGE SPECIFICATION, 'Online! 4 May 1998 (1998-05-04), XP002143375 Retrieved from the Internet: <URL:HTTP://JAVA.SUN.COM/DOCS/BOOKS/JLS/UNLOADING-RATIONALE.HTML> 'retrieved on 2000-07-24! the whole document — -/—	1,4,7

[X] Further documents are listed in the continuation of box C. [ ] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
26 July 2000	09/08/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax (+31–70) 340–3016	Bijn, K

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Interr nal Application No
PCT/EP 99/10301

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	VENNERS, BILL: "Under The Hood: Java's Garbage Collected Heap" JAVAWORLD, 'Online! August 1996 (1996-08), XP002143401 Retrieved from the Internet: <URL:http://www.javaworld.com/javaworld/jw-08-1996/jw-08-gc_p.html> 'retrieved on 2000-07-25! the whole document	1-9
A	SUN MICROSYSTEMS, INC.: JAVA DEVELOPER CONNECTION TECH TIPS", 'Online! 3 September 1997 (1997-09-03), XP002143402 Retrieved from the Internet: <URL:HTTP://DEVELOPER.JAVA.SUN.COM/DEVELOPER/TECHTIPS/1997/TT0903.HTML#TIP2> 'retrieved on 2000-07-25! the whole document	1-9

Form PCT/ISA/210 (continuation of second sheet) (July 1992)